

每周工作汇报

| | | | | | |
|----|-----|------|----------|------|-----------|
| 姓名 | 侯宇轩 | 开始日期 | 2019.5.7 | 结束日期 | 2019.5.13 |
|----|-----|------|----------|------|-----------|

1. 本周任务与计划

1.1 研究任务

阅读蔡老师布置的论文：PDE-Net: Learning PDEs from Data，学习其中的方法，思考如何用其对 level-set 进行改进。

阅读任重老师布置的论文：In Situ Video Encoding of Floating-Point Volume Data Using Special-Purpose Hardware for a Posteriori Rendering and Analysis，为之后实现这篇论文的算法铺垫。

2. 本周工作概要

2.1 当前的进展

本周工作

一、目标：使用 PDE-net 学习 Level Set

正在尝试使用上周的星型数据，结果目前尚未跑出。

二、论文阅读：In Situ Video Encoding of Floating-Point Volume Data Using Special-Purpose Hardware for a Posteriori Rendering and Analysis

下面贴出我的论文翻译。

In Situ Video Encoding of Floating-Point Volume Data Using
Special-Purpose Hardware for a Posteriori Rendering and Analysis
Nick Leaf, Bob Miller and Kuan-Liu Ma

摘要

由于存储和 I/O 带宽限制，科学计算模拟通常只存储计算出的时间步的一小部分。以前的工作已经证明了浮点体数据的可压缩性，但这种压缩通常在计算复杂性和可实现的压缩比之间进行权衡。这表明在 GPU 上使用专用的视频编码硬件进行压缩是非常有潜力的，但在目前，有 GPU 配备的超级计算机如 Titan 中完全没有使用此技术。本文研究发现，有损编码允许以足够的质量输出更多的数据，以便进行后期渲染和分析。同时研究发现，由于专用硬件的存在，编码过程可以与通用计算进行并行计算处理。最后，我们证明了在分析过程中，这种编码的体数据在内存中解码成本较低，因此不需要将解压缩的体数据存储在磁盘上。

1. 介绍

Ken Batcher 半开玩笑地说，“超级计算机是一种将计算极限问题转化为 I/O 极限问题的设备。”精确计算的进展只增加了与数据存储和移动相关的挑战。单个时间步的大数据量会给有限的磁盘配额带来压力，而写出存储单个时间步所需的大量数据对模拟速度有重大影响。诸如突发缓冲区(burst buffer)之类的创新减轻了这些问题，但没有完全解决。通常的解决方案是每几百个时间步只把一个时间步写到磁盘上，而把其余的时间步丢弃。这会导致断断续续的、急促的后期(post-hoc)时变(time-varying)可视化，并可能完全丢失输出步之间临时出现的重要特征。

原位(in situ)技术是解决这些 I/O 限制的常用方法。他们允许科学家在全部数据被丢弃之前对每一个时间步进行分析和可视化。它们还可以指导模拟监控，并可能根据早期输出修改或重新启动模拟运行。然而，所有原位技术都受制于先验知识问题：任何原位过程都必须提前指定。原位分析通常可以实时调整，但之前丢弃的时间步的结果不会反映此类调整。此外，原位分析并不是免费的，它会花费计算周期和 I/O 带宽，科学家们不愿意牺牲这两个带宽。

原位压缩是解决先验知识问题的一种方法。压缩方法不需要像原位分析程序那样进行详细的、特定于数据的调整。相反，分析是在事后(post hoc)运行的，可以根据需要重新运行和调整。高效的压缩可以显著地减少由于写出额外的时间步而产生的 I/O 压力。然而，压缩并不是无须计算的，计算成本往往随压缩效率的增加而增大。同时，并非所有的压缩方法都同样适用于体数据；一种非常有效的编码方法如果只为一般的一维数据设计的，可能会错过二维和三维数据冗余。

我们研究了专用视频编码硬件在解决这些问题上的应用。视频编码是为二维+时间数据设计的，非常适合压缩三维体数据。视频编码的计算成本很高，但在许多情况下，它是由专门为任务设计的硬件支持的。大多数高端 GPU 都如此，包括橡树岭国家实验室的 Titan 超级计算机中使用的 GPU。据我们所知，目前还没有在 Titan 上使用视频编码硬件的项目，因此它代表了尚未开发的超级计算资源。由于 GPU 视频编码发生在专用硬件上，因此即使启用了 GPU 的模拟过程也可以与编码并行执行，运行时影响非常小。我们对使用视频编码方法进行原位体数据编码做出以下研究：

| |
|---|
| 1.对将 32 位浮点数据转换为视频编码所需的 YUV 格式的转换方法研究 |
| 2.对可调编码参数对输出质量和大小影响的研究 |
| 3.对 HPGMG-CUDA[27]标准集的原位案例研究，试图最小化压缩对运行时间影响。 |
| 4.对事后(post hoc)加载和解压的时点(timing)研究，用于证明输出的可用性。 |

我们对视频体数据编码的评估显示，每次迭代运行时平均增加 40 毫秒，最坏情况下压缩比为 100:1，并且具有足够的后期视觉分析质量。

2.相关工作

许多研究人员已经引入了专门的压缩方法,以便在比特率和计算性能方面更好地处理体数据[3,4,9,17,26,31,35]。早期的方法,例如由 Ibaba 等人介绍的方法。倾向于完全依赖线性预测方法,如 Lorenzo 预测器[14]和小波压缩技术[15]。这类技术对于体数据的可伸缩压缩是一个重要的早期进展,因为它们可以通过核外(out-of-core)技术实现,从而避免了大量的内存开销。

最新的体积压缩方法,如 Fout 和 Ma[12]所介绍的方法,在前面的方法基础上进行了扩展,并利用了多个应用于数据的线性预测因子,自适应地选择方法。还有许多其他压缩体数据的方法来减少 I/O 和存储大小。此外,在用户 GPU 上直接可视化这些压缩体数据也是一种众所周知的技术,Rodriguez 等人[5]对各种建筑物的几种压缩容积数据的检测方法及不同的损耗和损耗压缩技术进行了综述。

上述技术与我们的工作尤其相关,因为它们基于的压缩流程(pipeline)与视频压缩技术中可用的流程类似,这种技术的硬件实现的广泛可用性是我们工作的主要动机。但是,需要注意的是,体积压缩还有其他方法,如基于网格的技术[16]、分形压缩方法[8]和 Shafaat[30]引入的自适应粗化(coarsening)技术。在重要性较低的区域自适应粗化数据,由此使得这些多分辨率方法有高度并行性,可以显著降低存储需求。然而,我们更感兴趣的是在模拟时(in simulation time)保持数据的完整分辨率。

许多现有的无损压缩方法本质上是串行的,因此不适用于高度并行的结构,如生成大容量数据时常用的结构。Wang 等[33]介绍了两种使用 GPU 高效地并行压缩浮点数据的无损方法。在集群或超级计算机上压缩大容量数据的缺点是压缩数据太复杂,不能在 GPU 上实时解压缩,因此需要作为后处理或在 CPU 侧进行解压缩,这会阻碍压缩数据的实际使用。因此,一些体积压缩技术是为后期可视化而设计的[11,20]。此外,有许多方法可以提高大规模体积可视化的性能,包括空块(block)跳跃、自适应细节级别(level-of-detail)、光线引导技术等[6]。

Gamito 和 Dias 等人员对使用图像压缩方法对浮点数据进行编码[13]进行了探索。他们探索了使用 Jpeg 2000 规范第 10 部分压缩浮点数据的方法,并注意到特殊的浮点值(如 nan)必须单独处理,因为它们与附近的值没有很好的关联。Usevitch[32]通过提出对浮点数据的 Jpeg 2000 压缩的无损方法来扩展这项工作。这些研究人员证明,虽然指数位和高尾数位相对可压缩,但较低尾数位很难与它们关联(因此不能很好的压缩)。

一些应用程序,例如那些使用医疗数据的应用程序,不能接受由于有损压缩方法而产生的伪影[5]。这不是一个新问题,处理浮点数据无损压缩的技术已经被许多研究人员探索过,例如 Engelson 等人[10]。Lindstrom 和 Isenburg[21]致力于提高这些算法的速度和效率,并改进了这些算法在运行时开销不可接受或需要并行实现时的适用性[25]。对于单精度浮点数据不足的情况,Burtscher 扩展了这些技术以有效地支持双精度数据[7]。由于较新的 GPU 硬件支持无损视频编码,它可以很容易地用于无损编码,尽管专用的方法可以产生更好的压缩比。请注意,虽然之前的工作是在 GPU 上压缩体数据[23],但这项工作使用的是 GPGPU 技术,而不是我们提议的高性能专用视频编码硬件[34]。

随着模拟方法的扩展,可用存储和该数据的可用容量之间的不平衡逐渐增长,需要开发用于各种不同类型数据的压缩技术和预条件器(preconditioner)[19,28]。即使在较小的规模下,Yeo[36]和 Schneider[29]也解决了在没有大消耗解压缩的情况下对压缩的体积数据进行解压缩、分析或呈现的问题。我们认为,通过使用视频压缩方法,我们可以利用广泛可用的解压硬件以及现有的数据块解压技术[24]来消除客户端解压的一些缺点,同时允许对现有的渲染器进行最小的更改,因为它们可以继续使用未压缩的规则网格数据。

研究人员还将视频压缩方法扩展到大型时变数据，如 Ko 等人[18]。Ko 将视频压缩方法扩展到 3D 帧，并提供了有效的压缩数据解压和渲染方法，支持针对其方法的细节级别 (level-of-detail) 和兴趣区域 (ROI) 选择。

3. 背景

尽管我们将编码器视为这项工作的黑匣子，但对它的一些了解对于指导分析和解释结果是有用的。目前最常见的两种视频编码方法 H.264 和 H.265（又称高级视频编码（AVC）和高效视频编码（HEVC））基本上使用相同的编码流程。H.265 需要更复杂的编码和解码，但在一半的比特率（即文件大小）下产生的视频质量大致相同。两者的编码流程如图 1 所示。第一阶段，浮点转换，是我们添加的一个附加阶段，用于将浮点数据转换为适合编码的像素数据，而不是普通视频编码流程的一部分。

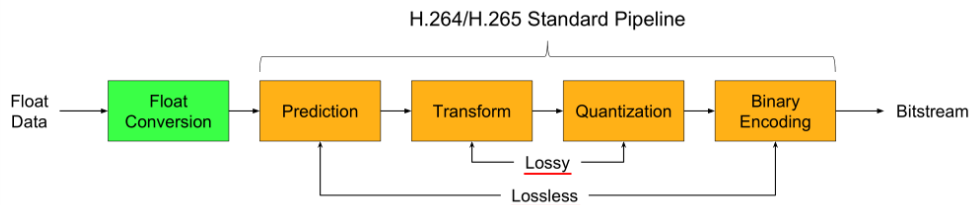


Figure 1: encoding pipeline. The *float conversion* stage converts the floats into a suitable YUV format. The remainder of the pipeline is unmodified. The *prediction* and *binary encoding* stages are both lossless. The *transform* stage imposes some loss due to conversion to frequency space. *Quantization* reduces the data range of the residual according to the user-specified QP parameter. The degree of loss from *Float conversion* depends on the chosen conversion method.

视频编码器处理像素数据，有时是 RGB 格式，但更典型的是 YUV。YUV 空间的一个主要优点是，人类视觉对颜色空间中的高频比亮度中的高频更不敏感。视频编码器经常通过使用下采样的 U 和 V 平面来利用这一事实。对于 H.264 和 H.265，YUV 4:2:0 是最广泛支持的颜色格式，下采样在水平和垂直尺寸上都是两个颜色平面的一半。因此，YUV 4:2:0 中的图像大约是 YUV 4:4:4 中一个图像的一半大小，其中所有组件都以相同的速率采样。软件编码器和更新的硬件编码器倾向于支持 YUV 4:4:4，但较旧的 GPU（如 Titan 中的 Tesla K20X）只支持 YUV 4:2:0。图像的每一个平面通常都有 8 位精度，尽管最新的 Nvidia GPU 和一些软件编码器支持 10 位平面。10 位支持通常必须在编译时为软件编码器配置，并且可能与 8 位支持互斥。

编码器通常每次提取单帧或单个二维切片的像素，并为每帧输出一个编码的数据包，同时有填充内部缓冲区的潜在延迟。每个帧被进一步划分为空间上相邻的像素块。流程的第一阶段，预测，利用相邻块之间的冗余。它确定编码器输出的帧类型。I 帧是内部编码的，这意味着预测只考虑同一图像中的块。P 帧和 B 帧可以混合使用帧间和帧内预测，其中 P 帧仅限于先前帧，B 帧可以按照显示顺序在其前后使用帧。对一个块的预测将一组操作应用于该块的邻域，并选择使预测值与原始数据（称为残差）之间的差异最小化的操作。由于残差通过流程被保存和转发，因此预测阶段是无损的。

变换阶段对块的残差应用离散余弦变换（DCT），转换到频率空间允许基于视觉显著性对细节进行优先级排序。变换阶段的损失是由于数值精度的限制造成的；之后，将频率空间剩余块输入量化级，根据量化参数（QP）减小频率分量的范围。量化是流程中最主要的信息丢失原因，而 QP 是在质量和输出大小之间进行调整的主要控制参数。视频编码通常由一些最大比特率目标控制。在可变比特率模式下，编码器将改变 QP 以满足用户定义的比特率目标。为了简化，我们只考虑使用恒定的 QP 模式来简化我们的评估。

最后，将像素块及其相关参数（量化范围、预测模式等）发送到编码器。编码应用无损压缩方法，如运行长度(run length)或算术编码。所产生的压缩位与一个头(header)相结合，生成一个 H.264 或 H.265 包，该包可以附加到文件中，也可以通过网络进行流式传

输。如果将原始 H.264/H.265 流作为后处理步骤混合为容器格式，则可以直接查看体数据视频。

4. 方法

由于这项工作涉及已经在硬件中实现的编码器，因此我们更关注它们的使用而不是它们的实现。我们将编码器视为一个黑盒，并检查输入和输出之间的关系。本节讨论了将浮点值转换为正确格式的多种方法。我们还详细介绍了我们在基准应用程序中的原位编码的实现，并展示了如何在加载时以较低的成本对编码数据进行解码，以便进行后期渲染。

4.1 将浮点数转换为 YUV 格式

我们考虑了一些将浮点转换为 YUV 的方法。此处对其进行了描述，并在第 5.1 节中进行了评估。因为我们关注的是体积可视化的最终目标，所以我们只考虑单精度 32 位浮点。我们还允许渲染考虑使视觉质量成为“足够好”的最终评判标准。

第一个被测试的转换方法，*componentbytes*，直接将 32 位浮点数格式中的 1 位符号位、8 位指数位和 23 位尾数位的“组件(components)”复制到两个 YUV 4:4:4 视频的通道中。由于该方法需要两个具有六个总通道的视频来编码所有的数据，所以我们选择将浮点数分成五个组成部分，而不是简单地在它们现有的顺序中取字节。直接转换的一个优点是它不需要计算或更改数据的范围，但是这种优点被它糟糕的压缩性能所抵消。最低的两个字节实际上是不可压缩的，这在之前的工作[12,13,32]中已经指出，并且与我们自己的结果一致（第 5.1.1 节）。特别是，编码器会将指数部分视为一组整数，其变化会在数据中创建人为的高频特性。

转换为整数格式允许我们更好地将数据与编码器设计处理的数字行为匹配。在应用编译器的默认浮点到整数转换之前，数据被规范化为范围 $[0 - (2^N - 1)]$ ，其中 N 是目标精度。由于没有明显的“最佳”精度可以采用，我们测试了 32、24、16 和 8 位无符号整数格式（分别命名为 *Int32*、*Int24*、*Int16* 和 *Int8*）。规范化可能是一个潜在问题，因为较大的数据范围可能隐藏某个子集内的细微行为。正如我们在 4.2 节讨论的，数据通常将编码为局部化块(*localized blocks*)，其中数据很可能位于总范围的一个子集内。局部化块可以单独归一化，在数据转换回浮点数进行可视化后，保留足够的精度以获得良好的视觉质量。

此外，可能需要特别考虑较大的转换格式。例如，24 位整数将适合于单个 YUV 4:4:4 视频，但 32 位整数必须在两个视频之间拆分。如果编码器仅支持 YUV 4:2:0，则需要在四个视频之间分割一个 32 位整数以保持所有字节的完全分辨率。这是由于需要色度下采样，此内容在第 3 节中出现。这可能会限制 NVENC API 允许的同时编码会话的数量，从而强制进行串行式编码。一种解决方案是将快速变化的、不太重要的字节放在全分辨率亮度平面中，将缓慢变化的、更重要的字节放在下采样色度平面中。然而，编码器将每个平面视为一个单独的维度；它们是相互独立编码的，并且每个平面将具有相似的误差量。最重要字节（*Most Significant Bit*, *MSB*，即最高位字节）的单个位中的错误将使其他字节相形见绌。10 位编码是这个问题的部分解决方案，但如第 3 节所述。其可用性有限。

颜色格式精度限制的一个潜在解决方案是交错两个或多个字节。由于每个 YUV 平面都是独立编码的，因此每个平面的精度限制为 8 位。如果最大平方误差(MSE，定义在第 5 节中)相当低，那么我们可以合理地假设误差仅限于每个平面的较不重要字节(*Less Significant Byte*)。*Int16interleaved* 方法通过交替分配位将 16 位整数拆分为两个字节。这将在 Y 平面和 U 平面的上半部分之间拆分最重要的字节，在下半部分中保留最不重要的字节(*Least-Significant Byte*，即最低位字节)。这在数学上相当于通过 YU 空间在 Z 型曲线(*Z-order curve*)上绘制。

4.2 原地编码(In Situ Encoding)

我们的原地编码器的目标是以完全分辨率、可接受的质量和模拟运行时间的最小可能影响为准则对数据进行编码。硬件编码实现使用专门为编码设计的专用、固定功能硬件引擎。因此，它可以与任何计算过程分开使用，也可以与任何计算过程并行使用。硬件编码几乎无处不在；它在 Intel 桌面处理器（快速同步视频）、AMD GPU（视频编码引擎）、NVIDIA GPU（NVENC）和其他处理器上提供。我们专注于 NVENC，以 Titan 主机的 Tesla K20X GPU 为目标。

为了正确测试数据是否可以与 GPGPU 计算同时编码，我们需要一个支持 GPU 的应用程序。我们选择了支持 CUDA 的 HPGMG-FV[2]基准(benchmark)，HPGMG-CUDA[27]。HPGMG-CUDA 是一个多网格微分方程求解器，它使用 OpenMP 和 CUDA 的组合来实现节点级的并行性。OpenMP 用于粗略的延迟限制级别，而 CUDA 用于精细的带宽限制级别。由于 HPGMG-CUDA 是一个基准代码，所以它已经对于大规模并行运行效果进行了全面的测试，并进行了优化以避免常见的瓶颈。混合计算提供了一个良好的对最坏情况的测试环境，它没有隐藏编码运行时影响的主要瓶颈。HPGMG 具有代表不同数量的多个矢量。我们只对解矢量进行编码。在实践中，模拟需要分别对每个期望的输出变量进行编码。

原地编码流程如图 2 所示。计算可以分为三个运行在两个线程中的块：求解(solve)、转换(convert)（线程 1）和编码(encode)（线程 2）。求解是未修改的多网格求解例程。转换阶段执行体数据的同时复制和转换为像素格式。最后，编码过程对数据进行编码并将其写入磁盘。solve 和 encode 是唯一两个可以并行运行的任务，因为 solve 和 convert 都访问原始数据，convert 和 encode 都使用转换后的数据。每个线程都是一个 OpenMP 任务，线程之间的同步是通过一个简单的互斥锁和脏位(dirty bit)来处理的。同步等待使用繁忙的等待循环，可能会浪费周期。我们通过在循环中放置一个 OpenMP 线程 yield 来缓解这一问题，它应该允许 OpenMP 重新捕获线程。



Figure 2: solve and encode overlap. The computation is broken into three parts: Solve, Convert (C), and Encode (Enc). Convert_n takes place within a lock section, and marks a dirty bit when it finishes. Encode_n and Solve_{n+1} can both start immediately afterwards. Encode_n waits for the lock, and for the dirty bit to be marked. Encode_n takes place within the lock, and clears the dirty bit before exit. Convert will wait for the previous Encode to finish before overwriting the converted volume, but this was not observed in practice.

由于我们只编码由 CUDA 处理的最精细的网格级别，所以 convert 不需要主机和设备之间的任何内存副本。convert 为所有体数据运行一个 CUDA 内核调用，该调用将每个元素转换为其 YUV 表示，然后再写入单独的体存储。如果模拟在覆盖体数据之前对体数据执行了足够的其他只读工作，那么可以在 encode 期间按需执行转换和复制，从而避免了进行复制的必要性。但是，HPGMG-CUDA 直接进入下一个 solve 过程，在这个过程中修改原始体数据。

一旦体数据被转换并标记为脏(dirty)，互斥锁被释放，下一个 solve 和 encode 阶段就可以继续了。编码器逐帧运行。创建编码器资源时，将分配若干个输入缓冲区（每个缓冲区包含一帧像素数据）和用于编码比特流的输出缓冲区。必须将体数据中的切片复制到要提交的编码器输入缓冲区。新版本的 NVENC 可以与 CUDA 进行共用，并直接将 CUDA 资源作为输入，但我们在编写时无法在 Titan 上使用此功能。输入帧被提交到编码器，直到进程用完空缓冲区。然后它处理任何可用的输出以释放输入缓冲区，并提交另一个帧。一旦提交了每个体数据切片，就会提交一个刷新代码，并处理所有剩余的编码输出，直到编

码器发出流结束的信号。编码输出在直接输出到磁盘之前被附加到主机端缓冲区。转换后的体数据脏位被标记为干净，互斥锁被释放，等待可能的 convert 过程。

HPGMG 将数据分配到大小相等的立方体“框”中，这些“框”尽可能均匀地分布在所有 MPI 级别(rank)中。通常，每个 MPI 级别将分配多个框。由于每个级别独立于其他级别进行编码和输出，因此在编码期间不需要通信。每个级别的框可以作为单独的编码流输出，但这会产生不必要的开销。我们选择将一个列中的所有框组合成一个视频。我们选择了二维布局框；这会增加单个帧的大小，从而为编码器提供更多潜在的计算并行性和信息冗余。它还减少了提交的帧数（与单独的流相比），从而减少了编码所需的 CPU 工作量。我们对这些盒子使用了一个简单的 Z 顺序布局。具有奇数个框（经常出现）的级别在帧中会有空白空间，但实际上这不是问题，因为空白的空间压缩得很小。

4.3 事后(post hoc)解码和可视化

原位压缩方法的一个优点是减少了编码体数据的存储影响。如果在加载之前必须将体解压缩到磁盘上，那么这个好处将大大降低。幸运的是，解码速度足以让体数据在加载时快速解码。我们通过在加载后立即添加解码阶段来修改用于并行体绘制的标准流程。一旦体数据块被解码，这个过程就如同从一开始就使用原始体数据一样。大多数可视化集群都配备了硬件解码器的 GPU，但我们发现在实际应用中，软件解码速度足够快，因为在加载过程中解码只发生一次。

加载压缩数据时的一个限制是每个编码的块必须全部加载和解压缩。如果运行模拟的机器的各个节点比渲染机器的节点更强大或内存更大，则这是一个问题。然而，有一些潜在的解决办法。离线预处理步骤可用于解码、细分和重新编码任何过大的编码块。这个过程也可以通过修改数据分发阶段在每个切片的基础上工作来实时(on-the-fly)完成。在实践中，与模拟机(simulation machine)相比，渲染机(rendering machine)可能拥有较少的单独的、功能更强大的节点，因此每个渲染节点将加载多个编码的块。

5 评价

我们的评估可以分为三大部分：编码参数的研究、现场性能测试，以及显示压缩数据可以在加载时快速解压缩。由于我们的最终目标是体绘制，因此我们严重依赖定性视觉结果来评估质量，尤其是与输出大小进行比较。我们还测量峰值信噪比（PSNR）作为质量的定量度量，我们将在下面描述。我们测量了现场编码和加载时间解码的时间性能。

PSNR 是无损压缩和信号处理文献中测量信号质量的标准统计。设 V 为 $S=w \times h \times 1$ 的规则网格体数据，值在[0-1]范围内。 E 是经过编码和解码后的同一体， V_i 和 E_i 是这两个体数据中的对应元素。PSNR 是使用两个体数据之间的均方误差（MSE）计算的，数据要相对于最大值（记为 max）进行缩放(scaling)。由于在转换和编码之前对体数据进行了规范化，因此在所有情况下 $\max=1$ 。MSE 和 PSNR 定义如下。

$$MSE(V,E) = \frac{1}{S} \sum_i^S (V_i - E_i)^2 \quad (1)$$

$$\begin{aligned} PSNR(V,E) &= 20 \log_{10}(MAX) - 10 \log_{10}(MSE(V,E)) \\ &= -10 \log_{10}(MSE(V,E)) \end{aligned} \quad (2)$$

在视频和图像压缩中，PSNR 与视觉质量有很好的相关性，但对于浮点体数据（见图 8 和图 6e），PSNR 的问题更大。此外，由于实际中使用的传输函数往往是非线性的，因此较小的压缩误差会导致较大的视觉差异。尽管存在这些缺点，但 PSNR 是一个有用的统计指标，我们发现它通常与比较相同体积数据的不同编码结果有关。

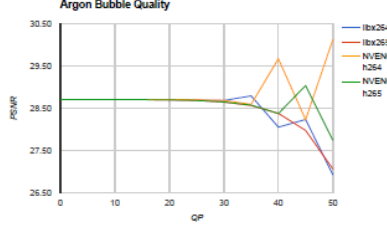


Figure 8: an example of unpredictable PSNR behavior. The PSNR for the Argon Bubble data using *int8* varies wildly for QP values above 30. Based on examination of the qualitative results, the outlier PSNR values near the end of the range are certainly not indicative of higher quality. This highlights the limitations of the use of PSNR as a quantitative metric for encoding floating point data.

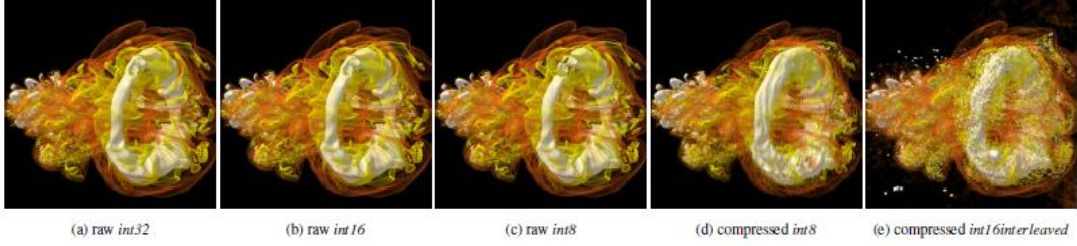


Figure 6: Argon Bubble conversion comparison. (a), (b), and (c) were taken after conversion to integer and directly back, with no encoding. (d) is *int8* encoded with QP=15 using libx264. Encoding higher precisions offer the same or, in some cases, worse visual quality. (e) shows the results using interleaving, which places alternating bits in order into the Y and U bytes. Note the significant visual noise from artificial high frequencies.

5.1 编码参数研究

在单个节点上使用专门构建的应用程序进行参数研究。对于每个参数组合，应用程序加载体数据，将其转换为可编码的表示形式，对其进行编码，将其写入磁盘，然后将其读回，并使用原始体数据计算差异统计信息。或者，可以对编码的体数据进行截屏。为了公平地比较图像，选择了一组渲染参数（例如视图参数、一维传递函数）来渲染每个体数据。手工创建传递函数自然会引入一定程度的主观性。我们试图创建传递函数，它在原始数据中显示了有趣的特性，并且不考虑压缩可能对它们产生的伪影。一般来说，使用高频传递函数时，压缩伪影最为明显，但在实践中，这种传递函数通常是可取的。

我们利用七个跨多个应用领域的规则网格体数据集进行参数研究。它们与一些统计数据一起列在表 1 中。每个数据集的 Ground Truth 图像如图 3 所示。Argon Bubble、NCAR Plume 和 JHTDB QCR 数据集均来自计算流体动力学模拟。JHTDB QCR 使用一个导出的统计方法，即 Q-准则，这有助于可视化涡流。The SuperNova 数据集来自天体物理模拟，The Visible Female 数据集来自扫描。Random 数据由每个体积元素的每个元素伪随机浮点数组成，这对于比较非常有用，但是对于这种情况不可能有有意义的 Ground Truth 图像。

Marschner-Lobb [22] 是一种综合基准数据集，通常用于评估体积渲染。

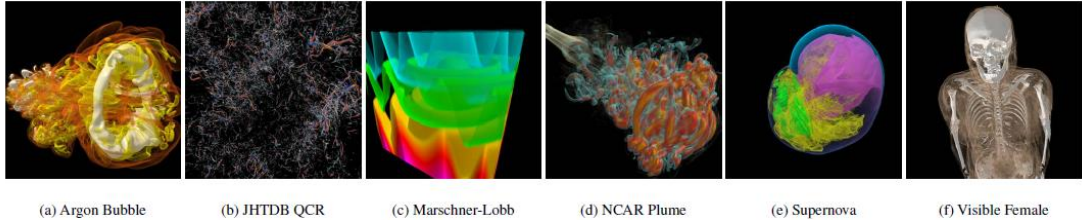


Figure 3: ground truth images. Five real and one synthetic (Marschner-Lobb [22]) datasets were used. These images show the original, unmodified data visualized directly with a hand-made transfer function. The exact same transfer function was also used to visualize the data after encoding and decoding.

5.1.1 压缩

压缩性研究旨在深入了解视频编码器如何处理 32 位值的不同组件(component)。每个编码的组件被写入一个单独视频文件的亮度平面，并记录产生的编码文件大小。图 4 显示了每个组件文件的大小，以组件字节和 Int32 格式的原始体数据的百分比表示。显示的结果来自使用 libx264 的编码；我们还使用 libx265 测试了压缩性，但是结果非常相似，以至于我们选择忽略它们。我们选择使用 0 的 QP 来避免纯粹由于量化造成的任何压缩。在两种格式转换之前，原始浮点值被规范化为[0 - 1]。由于输入数据不包含负数，所以我们选择不显示浮点符号位的可压缩性。

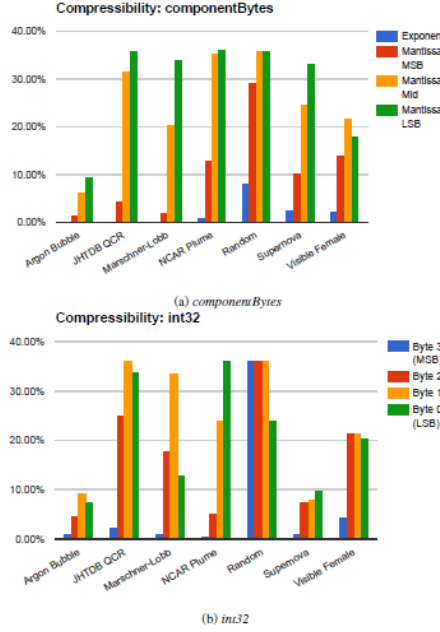


Figure 4: compressibility of components. Each byte of the 32-bit output was compressed independently using libx264 with a QP of 0. The resulting file sizes are shown as a percentage of the size of the original data. The values were separated by float component in Fig. 4a, and by byte after conversion to integer in Fig. 4b. Any results above 25% indicate compressed components that were larger than the original data. For most datasets, the two least significant bytes are effectively incompressible using video encoding.

componentbytes 的压缩性结果（图 4a）证实了先前文献[12, 13, 32]的发现：尾数的最低字节（LSB）和中间字节的熵太高，无法有效压缩。我们可以看到，组件通常大于原始大小的 25%，这意味着编码实际上增加了数据大小。尤其是 LSB 对大多数数据集的压缩效果并不比随机数据好。我们可以看到，在大多数情况下，尾数的最高字节（MSB）可以有效地编码。Random 数据集是可以预见的例外，只有随机数的最高有效字节比其他两个字节编码得更好，因为尾数的最高有效位只有七位。我们在 The Visible Female 数据集中观察到一个奇怪的模式，其中 LSB 编码比中间字节好。这可能是由于数据被从另一种格式重新采样成浮点数，导致 LSB 中出现一些偏移。我们还观察到 Argon Bubble 数据似乎在 componentbytes 中压缩得相当好。

Int32 的结果（图 4b）显示了不同的模式。在大多数情况下，LSB 的编码效率比中间字节高。通常，这种差异是相当小的。在 Marschner-Lobb[22]数据的情况下，很大的差异很可能是由于在体积采样位置之间存在相当大比例的值的混叠造成的。有趣的是，The SuperNova 的所有字节在 Int32 中的编码都比在 ComponentBytes 中好得多。Argon Bubble 数据在这里再次压缩得很好。很难猜测是什么导致了这一切，但 5.1.2 章表明原因似乎不会影响转换数据的视觉质量。

5.1.2 整数转换的影响

图 5 量化了转换为整数格式的效果。图 5a 显示了没有任何编码的每种整数转换类型的 PSNR。体数据被规范化，转换为给定的整数格式，然后直接转换回浮点数，并使用转换前后体积计算 MSE。我们可以看到一个在数据集之间保持的规则模式，在这个模式中，降低一个字节的精度会导致 PSNR 减少 25%—30%。请注意，Argon Bubble 的 Int32 转换和 JHTDB QCR 是完美的，它们的前转换值和后转换值是相同的，这意味着 PSNR 是未定义的。为了提供一个值，我们使用单个体素最大舍入误差替换了 MSE。

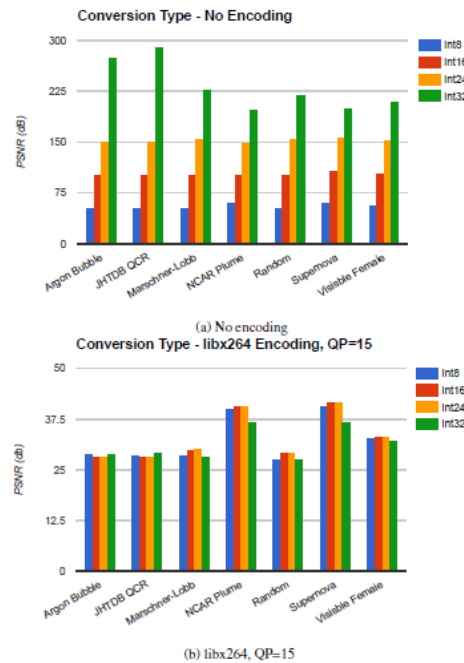


Figure 5: PSNR for different conversion methods. Fig. 5a shows the PSNR from converting to the given integer format without any encoding. Each byte of precision sacrificed results in a regular, predictable drop in PSNR. With encoding Fig. 5b, however, the PSNR is relatively flat across all precisions. Since video encoding handles each byte independently, any loss in the most significant byte dominate the output, effectively capping the PSNR.

PSNR 值的表现非常不同，即使损失很小。图 5b 显示了 QP 为 15 的转换和编码结果。尽管有些数据集的损失比其他数据集小，但每个数据集的 PSNR 值在转换器之间是相似的。这很可能是因为 MSB 中的任何错误都将主导结果。实际上，带编码的 Int32 的 PSNR 值与不带编码的 Int8 的值类似。

转换测试的图像确认主配电板中的错误占主导地位。图 6 显示了四种整数转换类型中每种类型的 Argon Bubble 图像，这两种转换类型都没有任何编码，并且在使用 libx264 编码之后，QP 为 15。我们可以看到，Int8 产生了一些可见的损失，但 Int16 和 Int24 与 Ground Truth 是不可区分的。所有图像在按预期编码后都显示出与 Ground Truth 的明显差异。后编码图像似乎使用较低的精度转换提高了质量。由于提高的精度在编码后不能提供更好的图像，所以我们选择只使用 Int8 进行进一步的测试。

Int16interleaved 的转换结果分别显示在图 6e 中。位交错至少在用于填充同一图像的两个平面时，会产生 PSNR 的边际增加（对于常规 Int16，为 31.5 对 28.4），但会产生明显的视觉噪声。最有可能的解释是交错将最高位有效位的低有效端推入 y 平面字节的上半部分。由于这些比特在更高的频率下发生变化，它会人为地放大高频。视频编码主要针对视觉显著性进行调整，特别是保持高频、高振幅的效果。因此，这种人工噪声更受编码器的青睐和保护。仅使用两个单独图像的 Y 平面会产生更大的视觉噪声，这意味着亮度和色度平面是分开处理的，即使对于 YUV 4:4:4。

5.1.3 量化参数的影响

控制 QP 对于调整输出质量与大小非常重要。例如，libx264 和 libx265 通常建议的默认 QPs 分别为 23 和 28。然而，对于视频数据来说，那些很好的建议并不一定能转化为体积渲染。我们使用一系列的 QP 值记录了每个数据集和编码器的定量（PSNR 和压缩比）和定性结果。The SuperNova 数据集的结果如图 7 所示。在 QP=25 左右之前，PSNR 相对平坦，之后开始四次下降。这反映了定性结果，其中至少 QP=15 的所有图像在很大程度上是不可区分的。在 QP=20 时，NVENC H.264 编码器与图 3e 中的 Ground Truth 图像相比，会产生一些退化，并且伪影会在 QP=40 时变得相当严重。

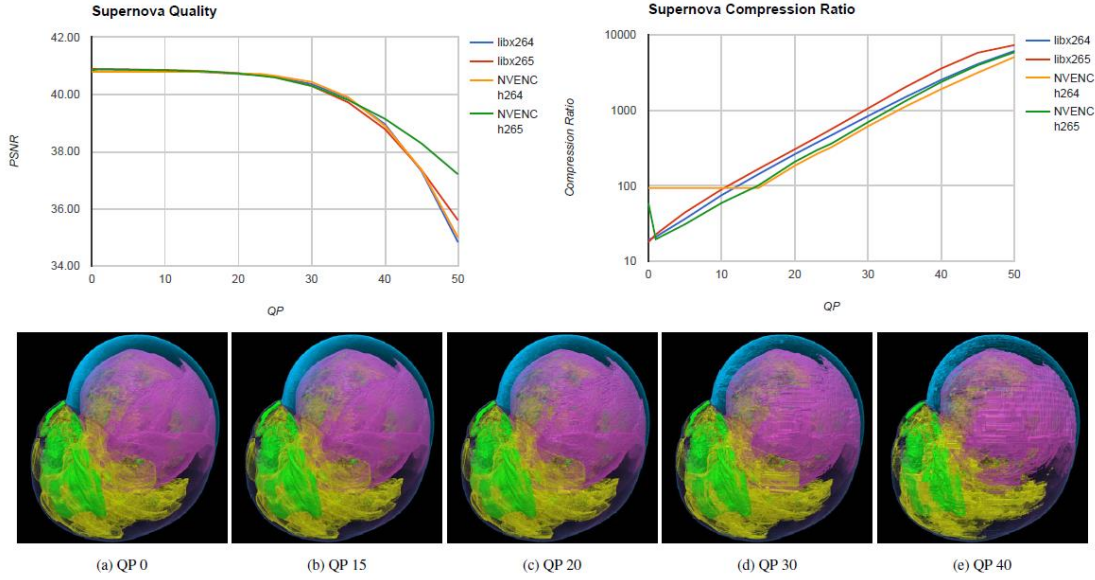


Figure 7: effect of varying QP on Supernova. PSNR values remain relatively flat until around QP=25, but fall afterwards. Compression ratios mostly grow exponentially throughout (the abnormalities with NVENC encoders are discussed in Sect. 5.1.3). The qualitative results using the NVENC H.264 encoder show slight artifacts at QP=20 which become pronounced by QP=30. The QP behavior suggest a QP of 15 as a baseline value for NVENC H.264.

图 7 所示压缩比的对数图显示了质量与尺寸之间的另一半权衡。压缩比随着 QP 的增加呈指数级增长，导致文件大小越来越小。NVENC 编码器在 QP 值小于 15 的情况下表现出一些奇怪的行为。NVENC H.264 编码器似乎将小于 15 的任何值的 QP 替换为 15。NVENC H.265 编码器在 QP=0 时的行为不同，这可能表明编码器检测并更改该特定值的操作。所有测试数据集的压缩大小都有类似的模式，尽管一些数据集似乎渐进地接近高 QP 值时最大压缩比的限制。这个限制很可能是由作为编码输出一部分的元数据施加的开销决定的。

对于大多数数据集，定量 QP 曲线是相似的，但偏移量不同。例如，The SuperNova 数据集的基线 PSNR-PSNR 在 QP=0 至 15 范围内偏高，但在相同范围内压缩比最低。然而，也有一些特定的异常值。图 8 显示了 Argon Bubble 数据集的 PSNR 与 QP 曲线。虽然 libx265 曲线看起来像预期的那样，但对于超过 30 的 QP 值，其他编码器都显示出奇怪的不可预测行为。我们观察到与 JHTDB 数据集的曲线类似的模式。使用 NVENC H.264 编码器在 QP=40 下对 Argon Bubble 数据集进行定性分析，无法解释为什么它的 PSNR 特别高。现在，我们对这种奇怪的行为没有任何解释。

5.1.4 体分区大小

如第 4.2 节所述，每个 MPI 等级输出其负责的任何本地数据。这意味着输出大小可能会因应用程序和模拟配置的不同而发生很大的变化。图 9 显示了我们的分区大小研究的编码结果。我们使用了 512³ Marschner Lobb[22]数据集，之前的测试显示，它具有相当具有

代表性的编码性能，而且我们知道它没有完全一致的区域。测试将 512^3 均匀地划分成一组大小相同的立方块，并使用四个编码器中的每一个单独对每个块进行编码。编码文件的大小被求和，并以原始大小的百分比显示。在特定编码器无法使用给定的块大小运行时，缺少某些条目。所有块大小都用相同的 QP (15) 编码。

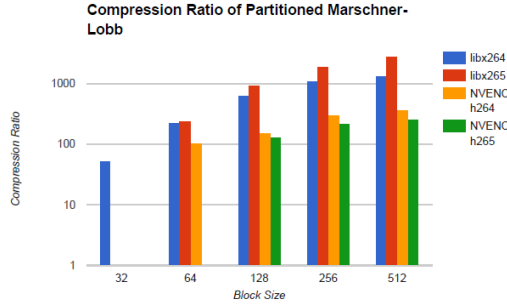


Figure 9: compression ratio using varying block sizes. The volume was divided into even size 32^3 to 512^3 blocks, and each block was converted (*int8*), encoded, and output separately. Smaller block sizes drastically reduce compression ratios—there is a factor of 25 difference in compression ratio between 32^3 and 512^3 blocks for the libx264 encoder. Data is missing where the given encoder failed to encode the given block size.

块大小和压缩效率之间存在明显的正相关关系。编码器可用的数据越多，它可以利用的数据冗余度就越大。这种效果会受到收益递减的影响，但似乎任何大于 256^3 的代码都应该能够很好地编码。另一方面， 32^3 块的编码效率较差。依赖大量小容量块的模拟可能需要在编码前收集数据。这种收集通常可以在同一节点上的级别之间本地完成，从而绕过交换和聚合块进行编码的通信成本。

5.2 原位方法性能

我们在橡树岭国家实验室的 Titan 超级计算机上进行了现场性能研究。我们的运行配置反映了最近的 HPGMG 性能结果[1]，每个物理节点有一个级别(rank)，（即每个 GPU 有一个级别），每个列有 4 个 OpenMP 线程，在最好的网格级别上每个列有 8 个 128^3 体盒。图 10 中弱标度测试的结果以每秒自由度 (DOF) 表示，该自由度是指每个模拟顶点的组合标量 DOF。我们测试了 1 到 4096 个节点，不过由于一个未解决的内存错误，我们对 2048 个节点使用了一个插值值。总的来说，基准测试在没有编码的情况下性能稍好，但对所有测试过的机器大小的影响都很小。每次迭代运行时间的平均差异为 40 毫秒。由于 HPGMG 只代理模拟的解算器部分，因此完整的应用程序还将执行其他工作。这个额外的每次迭代时间为隐藏编码和输出延迟提供了更多的机会，并且应该导致更低比例的运行时间损失。

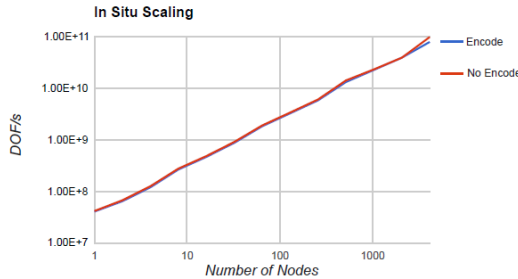


Figure 10: weak scaling performance with and without in situ encoding in Degrees of Freedom (DOF) per second. Since encoding runs in parallel with the next iteration step and does not require any CUDA resources, it has a marginal overall impact on performance. Conversion (using *int8*) requires write-protected access to the data, but takes very little time.

5.3 临时加载和解码

为了检验编码的体积数据的可用性，进行了事后加载试验。理想情况下，数据应该保留在磁盘上编码，并且只有在为可视化而加载数据时才能动态解码。图 11 显示了 Argonne 国家实验室的 Cooley 可视化集群上的弱缩放加载时间。我们测试了 1 到 64 个 MPI 等级，每个物理节点有 2 个等级，每个 GPU 核心有一个等级。体大小被缩放以保持每 MPI 列 512^3 个体素。我们将只加载原始数据的时间与加载压缩体块、对其进行解码并将其转换回浮点数据进行渲染的时间进行了比较。由于这两种类型的数据在解码后处理是相同的，因此我们省略了后期渲染流程中的任何计时结果。

从图 11 中，我们可以看到编码的数据在所有机器尺寸上都有一个相当平坦的加载时间。合成的 Marschner-Lobb[22]数据被用来轻松控制数据大小，但如图 12 所示，它的压缩非常有效。加载时间很可能主要由解码时间决定，然后，每级解码时间是恒定的。相比之下，原始数据加载时间在小型机器上表现出色，在大型机器上落后。加载原始数据需要更多的网络带宽，但不需要任何每级别的处理时间，这可能是增加的原因。两个时间序列都显示出大量的可变性，根据我们的观察，这很可能是由于文件系统上的缓存造成的。

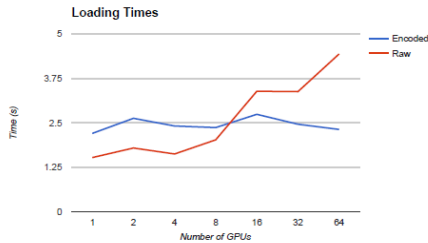


Figure 11: weak scaling comparison of load times for raw and encoded volume data. The measured time encapsulates both the load from disk into memory and, in the case of encoded data, decoding and conversion to floating point data. Each MPI rank handles a single 512^3 block, which can all be decoded in parallel, making the decode time constant for all numbers of nodes. Load times for small machine sizes are dominated by the per-block decode time, while the cost of loading raw data dominates for a larger number of nodes.

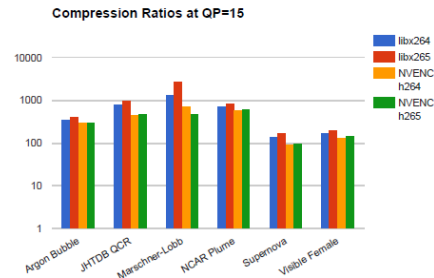


Figure 12: compression ratios at using *int8* at QP=15. There is significant variability in compression ratios across datasets. This is to be expected, as the degree of compressibility is dependent on the data. The Supernova data, encoded using the NVENC H.264 encoder, is the only data-encoder combination with a compression ratio less than 100:1.

6 讨论与未来工作

参数研究提出了以合理的成本进行高效编码的设置。对于将浮点数直接转换为整数的转换方法，在单独的颜色通道中编码多个字节没有任何好处。除非有更高精度的编码（10 位或 12 位），否则 *Int8* 是最佳选择。在未来，我们将探索将浮点数转换为 YUV 格式的其他方法。我们考虑的一种方法是基于柱状图的方法，在这种方法中，我们将更多的整数值分配给值范围中填充更多的部分，同时保持整体数据顺序。之后我们将不得不将这个柱状图存储为元数据，但是与数据大小相比，这样的柱状图可能很小，并且可以使用一个简单的二进制编码方法来进一步减小它的大小。对 QP 的研究表明，在大多数情况下，15 或更低的 QP 应足以满足至少 100:1 的压缩比（图 12）。分区大小研究表明编码器应该始终使用本地节点上的所有可用数据，但是 128^3 块应该提供一个合理的下限。

原位编码的主要考虑因素是：1. 变量选择。2. 选择质量与输出尺寸权衡点。3. 获得足够大的块大小。4. 最大化编码和计算之间的重叠。由于范围考虑，变量选择很重要。某些变量通过转换为 *Int8* 而比其他变量编码得更好，损失更少，一些变量（如 JHTDB 数据集的 QCR）处理编码损失比其他变量更差。所需的文件大小将取决于科学家选择写出数据的频率，而在大多数情况下，QP 15 工作得很好。较低的 QP 通常是可取的，但可能取决于采用 H.265 编码的较新硬件，具体取决于实现情况。在大多数情况下，可能可以实现足够的块大小，但可能需要将某些级别作为编码级别来专用，这取决于并行化策略；对于 HPGMG 来说，这不是必需的，但是每个列有少量数据的模拟可能需要它。计算和编码重叠可以通

过简单的基于任务的并行实现，但执行可能会根据上下文略有不同。如第 4.2 节所述，如果有一段时间可以保证他们不会写入数据，那么一些模拟可能能够避免创建数据的副本。否则，需要单独的副本进行编码。

由于我们的编码是有损的，因此该技术的输出可以补充并可能降低完整数据写入的频率，但不能完全消除它们。压缩的体积也可能需要保持时间连续性。完整的原始数据与压缩数据之间有足够的差异，可能会导致动画期间某些视觉功能“弹出”。如果需要，可以在事后创建完整数据的编码副本。

Nvidia 编码 API 提供的一个有趣的特性是运动估计（Motion Estimation, ME）模式。在这种模式下，API 从预测阶段返回 ME 向量，这通常是一个计算密集的过程。他们的文档表明，这可以用来加速 NVENC 不支持的自定义编码器，例如，ME 矢量可以用来加速特定浮点数的预测方法，如[12]。ME 向量也可以完全用于单独的目的，例如特征分割或跟踪。仅限 Me 模式仅在 NVENC 版本 6 和更高版本的文档中提到，在较旧的硬件上可能不可用。

7 结论

视频编码确实对体数据有用，浮点值可以编码为 YUV 格式，其质量足以进行大多数视觉分析。通过使用大多数 GPU 上已经存在的专用硬件，这种编码计算非常容易。参数研究发现，压缩比通常优于 100:1，从而导致文件非常小，并使模拟输出具有更高的时间分辨率。在 HPGMG-CUDA 基准测试上进行的案例研究表明，这种编码可以在对运行时影响很小的情况下就地进行，而在完整的模拟中，这种影响很可能忽略不计。最后，事后加载和解码研究表明，数据可以在加载时快速解码，无需对磁盘上的数据进行解码。

三、浙江省面上基金-超高清脑影像可视化

已经将时磊老师的内容整合进文档，删去了原来 method 部分照抄周昆老师基金的部分，完成了又一版内容。

3. 下周工作计划

等待星型数据的运行结果，进行调参。

参考 Encoding 论文的参考文献，对该论文进行深入理解，整理其实现需要的步骤。

附表：工作整理

| 任务类型 | 任务内容 | 截止日期 | 当前进度 |
|------|------|------|------|
|------|------|------|------|

| | | | |
|----|---------------------------------|--|---|
| 工作 | PDE-net 与 level set 的结合 | | <p>蔡老师提出新方法：使用偏微分方程网络 PDE-net 对 level set 进行改进。</p> <p>现在正在对数据进行测试。</p> |
| | 脑影像可视化基础：实现 In Situ Encoding 论文 | | 刚刚对论文完成翻译。 |

本周工作时长：8 小时*5 + 6 小时*2 = 52 小时。